http://www.nightwing.com.au/FileMaker/ CobaltSky@nightwing.com.au

FileMaker Pro™ AutoIndexing

Paper prepared by R J Cologon PhD

November 2002

Background

In FileMaker Pro, calculations which include references to global fields or related fields (ie fields from other databases) cannot be stored or indexed.

Indexes, however, perform several important functions:

1. they facilitate searching on a field
2. they enable the creation of value lists based on field contents
3. indexed fields can be used as the target (foreign) key in a relationship

In addition, though indexes are not used for sorting, stored values are nevertheless sorted more efficiently than unstored ones (since the calc does not have to be evaluated as the sort progresses).

Each of these considerations is significant. however the latter two sometimes present the database designer with a dilemma. Occasionally, the natural connections between data elements require that a relational dependency - or a global variable - be factored into the production of a key field or value list

This paper is concerned with techniques which may be used to ameliorate a situation in which a relationship of value list cannot be created because the field on which it is to be based cannot be indexed.

The Scripted Solution

One answer to this problem lies in the use of a script rather than a calculation.

Whereas a calculation determines a result based on a formula, a script is able to place a calculated result - using an equivalent formula - into a text or number field. A result so placed may be indexed as data, regardless of whether its components are local, global or related.

While a scripted procedure to write a calculated value to a data field throughout the current file can be set up using the 'Replace Contents' (formerly 'Replace') script step, this will not be suitable for use in a multi-user solution, it will skip over any records currently being edited by another user in multi-user mode, without producing an error code or any means to trace the records which have not been updated.

A better alternative, therefore is a loop which passes through the file applying the calculation to each record in turn, via a Set Field [ ] command. Immediately after each Set Field step, error-trapping can be used to ensure that the record was not locked, and an alternate procedure can be invoked if it was (eg the record number can be stored, so that a subsequent pass can be made, and/or the problem can be reported to the user at the closure of the script sequence).

Since either approach (Replace Contents or Loop/Set Field) works on the current found set, it will be necessary to either:

- precede the relevant step/s with a 'Show All Records' step to that all records are included, or
- if it is desired to complete the procedure without disturbing the found set, to perform the procedure twice, with a 'Show Omitted' step between the two passes, and a further 'Show Omitted' step at the end of the script.

In this latter case, a test to ascertain whether there is a found set in place should precede the second sequence, along the lines of:

```
If [“Status(CurrentFoundCount) < Status(CurrentRecordCount)”]
  Show Omitted
  Replace Contents [“Your Field’, Replace with calculated result: “Your Formula Here”]
  Show Omitted
EndIf
```

(note that this is the single-user-only Replace Contents version of the script).

Using this approach, however, it will be necessary to run the script whenever it is desirable to refresh the indexed field values. This is a limitation, since it will not always be convenient or efficient for users to run the script, so the solution is not suitable for all situations.

In a few cases, it may be possible to solve this problem procedurally (eg to call the script whenever a user enters a layout where a relationship or value list based on the field is to be accessed). In other situations, the solution will be less than seamless.

Automation

A better approach in some instances may be to set the indexable field to update automatically whenever a relevant event occurs. This would be done by setting the indexable field as a look-up (to copy the contents of a calculating field via a self-join based on recordID).

The trigger for the lookup should be based on a stored value local to the current record which, when it changes, will cause a refreshing of the lookup of the calculated value. This can be something specific to the nature and purpose of the calculation (eg a local value associated with it) or something as general as the record modification time. Once a suitable event to trigger by has been selected, a primary key field for the trigger should be created. This can be called 'cTrigger.key', and should be defined with a formula along the lines of:

```
Case(IsEmpty(TriggerField), Status(CurrentRecordID), Status(CurrentRecordID))
```

In addition to this, the following will be required:

1. a stored calculating field (call it 'RecordID') of *number* type with the formula set to Status(CurrentRecordID)
2. a self-join relationship called 'StoredCalcUpdate' which matches cTrigger.key with RecordID.
3. a data field (ie of type text, number, date or time, as appropriate) which will be the indexable field upon which a relationship of value list is to be based

The data field referred to at 3 (above) should be defined as a lookup to copy the value of the pre-existing unstored calculation field, based on the 'StoredCalcUpdate' relationship.

With these procedures in place, the stored data field will be refreshed automatically each time the value you have chosen as the basis of your trigger changes (eg if it is a field which auto enters modification time, every time anything on the record changes).

Choice of Trigger/s for the Update

It is important to recognise that this automation technique has a limitation insofar as it will only successfully trigger in response to changes which are local to the current record. This is because the table of field dependencies which FileMaker maintains in the background to ensure that dependent calculation fields are updated, is local to each file.

So, for example, if the trigger field references a global value, only the current record will update when that value is changed. If the trigger field references a value in another file (via a relationship), the triggering may be limited to changes made to the local value from a field placed on a layout in the current record.

For this reason it is preferable that a value be chosen as the trigger for the update, which is either:

- closely associated with the calculation function and therefore likely to change whenever there is a corresponding need to update the calculation value, or
- general to the record and likely to force frequent updates to ensure that the calculation is refreshed with maximum frequency.

For this latter purpose, a generic record-related value such as a record modification time field may be the best choice as the point of reference for the trigger.

A Combined Approach

The table of field dependencies that FileMaker maintains in the background and uses to track cascading recalculations is confined to stored calculations. Therefore it is important to note that although the approach can work very well in some instances, it is theoretically imperfect in that a value in another file can change remotely and this will not necessarily trigger the relookup.

In implementations where this limitation is of concern, a combined approach which includes a refresh script (eg based on a Replace Contents script step) as an adjunct to the triggering technique is a possible solution. This can either be triggered periodically by users or run routinely at start-up. shut down or linked to some other action such as change of layouts.

In cases where data on the current records is changing frequently, the added measure is unlikely to be necessary. However some combination of the two techniques outlined above is likely to provide an adequate solution for indexing problems in most situations.

Prepared by:
R J Cologon, PhD
NightWing Enterprises
Melbourne, Australia
CobaltSky@nightwing.com.au
http://www.nightwing.com.au/FileMaker/